Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de ciencias y sistemas

Organización de Lenguajes y Compiladores 2

Guía Técnica

Herberth Guillermo Obregon Espino

Carne 201314237

Guatemala, lunes, 22 de junio de 2020

Introducción

Este es un manual el cual comprende de una aplicación que sea capaz de ejecutar codigo a partir de un lenguaje formal dado, permitiendo la utilización de operaciones aritméticas de suma, resta, multiplicación y división entre números enteros y decimales.

Clases y métodos

Clases y Archivos:

Lex1.py

*from* compi.gramaticas.globales *import* \*  
  
tokens = (  
 **"RABS"**,  
 **"RMAIN"**,  
 **"RGOTO"**,  
 **"RREAD"**,  
 **"REXIT"**,  
 **"RPRINT"**,  
 **"RUNSET"**,  
 **"RINT"**,  
 **"RFLOAT"**,  
 **"RCHAR"**,  
 **"RARRAY"**,  
 **"RXOR"**,  
 **"RIF"**,  
 **"PARIZQ"**,  
 **"PARDER"**,  
 **"CORIZQ"**,  
 **"CORDER"**,  
 **"MAS"**,  
 **"MENOS"**,  
 **"POR"**,  
 **"DIVIDIDO"**,  
 **"MODULO"**,  
 **"RAND"**,  
 **"ROR"**,  
 **"RNOT"**,  
 **"RANDBIT"**,  
 **"RORBIT"**,  
 **"RNOTBIT"**,  
 **"RXORBIT"**,  
 **"SHIFTIZQ"**,  
 **"SHIFTDER"**,  
 **"IGUAL"**,  
 **"DIFERENTE"**,  
 **"MAYORIGUAL"**,  
 **"MENORIGUAL"**,  
 **"MAYOR"**,  
 **"MENOR"**,  
 **"DECIMAL"**,  
 **"ENTERO"**,  
 **"CADENA"**,  
 **"PTCOMA"**,  
 **"DOSPUNTOS"**,  
 **"ID"**,  
 **"ETIQUETA"**,  
 **"ASIG"**,  
)  
  
# Tokens  
t\_RABS = **r"abs"**t\_RMAIN = **r"main"**t\_RGOTO = **r"goto"**t\_RREAD = **r"read"**t\_REXIT = **r"exit"**t\_RUNSET = **r"unset"**t\_RPRINT = **r"print"**t\_RINT = **r"int"**t\_RFLOAT = **r"float"**t\_RCHAR = **r"char"**t\_RARRAY = **r"array"**t\_RXOR = **r"xor"**t\_RIF = **r"if"**t\_PARIZQ = **r"\("**t\_PARDER = **r"\)"**t\_CORIZQ = **r"\["**t\_CORDER = **r"\]"**t\_MAS = **r"\+"**t\_MENOS = **r"-"**t\_POR = **r"\\*"**t\_DIVIDIDO = **r"/"**t\_MODULO = **r"%"**t\_RAND = **r"&&"**t\_ROR = **r"\|\|"**t\_RNOT = **r"!"**t\_RNOTBIT = **r"~"**t\_RORBIT = **r"\|"**t\_RANDBIT = **r"&"**t\_RXORBIT = **r"\^"**t\_SHIFTIZQ = **r"<<"**t\_SHIFTDER = **r">>"**# t\_IGUAL = r"=="  
t\_DIFERENTE = **r"!="**t\_MAYORIGUAL = **r">="**t\_MENORIGUAL = **r"<="**t\_MAYOR = **r">"**t\_MENOR = **r"<"**t\_PTCOMA = **r";"**t\_DOSPUNTOS = **r":"**t\_ASIG = **r"\="***def* t\_IGUAL(*t*):  
 """=="""  
 *return t  
  
  
def* t\_DECIMAL(*t*):  
 r"\d+\.\d+"  
 *try*:  
 *t*.value = float(*t*.value)  
 *except* ValueError:  
 print(**"Floaat value too large %d"**, *t*.value)  
 *t*.value = 0  
 *return t  
  
  
def* t\_ENTERO(*t*):  
 r"\d+"  
 *try*:  
 *t*.value = int(*t*.value)  
 *except* ValueError:  
 print(**"Integer value too large %d"**, *t*.value)  
 *t*.value = 0  
 *return t  
  
  
def* t\_ID(*t*):  
 r"[$][a-zA-Z]+[0-9]\*"  
 # cosas como $t1  
 lEtiquetasAmbito.update({*t*.value: etiquetilla})  
 *return t  
  
  
def* t\_ETIQUETA(*t*):  
 r"[a-zA-Z\_][a-zA-Z\_0-9]\*"  
 *if t*.value == **"abs"**:  
 *t*.type = **"RABS"** *return t  
 elif t*.value == **"array"**:  
 *t*.type = **"RARRAY"** *return t  
 elif t*.value == **"print"**:  
 *t*.type = **"RPRINT"** *return t  
 elif t*.value == **"unset"**:  
 *t*.type = **"RUNSET"** *return t  
 elif t*.value == **"char"**:  
 *t*.type = **"RCHAR"** *return t  
 elif t*.value == **"int"**:  
 *t*.type = **"RINT"** *return t  
 elif t*.value == **"float"**:  
 *t*.type = **"RFLOAT"** *return t  
 elif t*.value == **"goto"**:  
 *t*.type = **"RGOTO"** *return t  
 elif t*.value == **"if"**:  
 *t*.type = **"RIF"** *return t  
 elif t*.value == **"xor"**:  
 *t*.type = **"RXOR"** *return t  
 elif t*.value == **"main"**:  
 guardar\_etiqueta(*t*.value)  
 *t*.type = **"RMAIN"** *return t  
 elif t*.value == **"exit"**:  
 *t*.type = **"REXIT"** *return t  
 elif t*.value == **"read"**:  
 *t*.type = **"RREAD"** *return t  
 else*:  
 guardar\_etiqueta(*t*.value)  
 *return t  
  
  
def* find\_column(*input*, *token*):  
 line\_start = *input*.rfind(*input*, 0, *token*.lexpos) + 1  
 *return* (*token*.lexpos - line\_start) + 1  
  
  
*def* t\_CADENA(*t*):  
 r"(\'|\").\*?(\"|\')"  
 *t*.value = *t*.value[1:-1] # remuevo las comillas  
 *return t*# Comentario simple // ...  
*def* t\_COMENTARIO\_SIMPLE(*t*):  
 r"\#.\*\n"  
 *t*.lexer.lineno += 1  
  
  
# Caracteres ignorados  
t\_ignore = **"** \t**"***def* t\_newline(*t*):  
 r"\n+"  
 *t*.lexer.lineno += *t*.value.count(**"**\n**"**)  
  
  
*def* t\_error(*t*):  
 descripcion = **"Caracter no reconocido por Agus: "** + *t*.value[0]  
 ErrorLexico = CError(descripcion, *t*.lexer.lineno, find\_column(entrada, *t*), **"lexico"**)  
 lErrores.append(ErrorLexico)  
 *t*.lexer.skip(1)

yacc1.py

*from* compi.gramaticas.lex1 *import* tokens  
*from* compi.helpers.ast *import* ASTNode, LexToken  
*from* compi.gramaticas.globales *import* \*  
  
  
*def* find\_column(*inp*: str, *token*: any):  
 line\_start = *inp*.rfind(*inp*, 0, *token*.lexpos) + 1  
 *return* (*token*.lexpos - line\_start) + 1  
  
  
# Asociación de operadores y precedencia  
precedence = (  
 (**"right"**, **"RNOT"**),  
 (**"left"**, **"RAND"**, **"ROR"**, **"ETIQUETA"**),  
 (**"left"**, **"IGUAL"**, **"DIFERENTE"**),  
 (**"left"**, **"MAYOR"**, **"MENOR"**),  
 (**"left"**, **"MAYORIGUAL"**, **"MENORIGUAL"**),  
 (**"right"**, **"RNOTBIT"**),  
 (**"left"**, **"RANDBIT"**, **"RORBIT"**, **"RXORBIT"**),  
 (**"left"**, **"SHIFTIZQ"**, **"SHIFTDER"**),  
 (**"left"**, **"MAS"**, **"MENOS"**),  
 (**"left"**, **"POR"**, **"DIVIDIDO"**),  
 (**"left"**, **"MODULO"**, **"RABS"**),  
 (**"right"**, **"UMENOS"**),  
)  
  
  
*def* p\_init(*t*):  
 """ initGram : RMAIN DOSPUNTOS instrucciones"""  
 init\_ast = ASTNode(**"start"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]), *None*, *t*[3])  
 init\_ast.childs = *t*[3]  
 *t*[0] = init\_ast  
  
  
*def* p\_instructions\_list(*t*):  
 """instrucciones : instrucciones instruccion"""  
 *t*[1].append(*t*[2])  
 *t*[0] = *t*[1]  
  
  
*def* p\_instructions\_list\_1(*t*):  
 """instrucciones : instruccion"""  
 *t*[0] = [*t*[1]]  
  
  
*def* p\_instructions\_eval(*t*):  
 """instruccion : declaracion\_registros PTCOMA  
 | funcion\_print PTCOMA  
 | funcion\_unset PTCOMA  
 | salto\_condicional PTCOMA  
 | instruccion\_control PTCOMA"""  
 *t*[0] = *t*[1]  
  
  
*def* p\_instruction\_exit(*t*):  
 """instruccion : REXIT PTCOMA"""  
 *t*[0] = ASTNode(*t*[1], *t*.lexer.lineno, find\_column(entrada, *t*.slice[2]))  
  
  
*def* p\_instructions\_error(*t*):  
 """instruccion : error PTCOMA"""  
 NodoSentencias = ASTNode(**"error"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[2]))  
 *t*[0] = NodoSentencias  
  
  
# print("Error sintáctico en '%s'" % t.value)  
  
  
*def* p\_instructions\_tag(*t*):  
 """instruccion : ETIQUETA DOSPUNTOS"""  
 *t*[0] = ASTNode(**"etiqueta"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]), *None*, ASTNode(*t*[1]))  
  
  
*def* p\_salto\_condicional(*t*):  
 """salto\_condicional : RGOTO ETIQUETA"""  
 NodoSaltoCond = ASTNode(  
 **"salto\_incondicional"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1])  
 )  
 NodoSaltoCond.add\_child(ASTNode(*t*[2]))  
 *t*[0] = NodoSaltoCond  
  
  
*def* p\_instruccion\_control(*t*):  
 """instruccion\_control : RIF PARIZQ expresion PARDER RGOTO ETIQUETA"""  
 NodoControl = ASTNode(  
 **"sentencia\_control"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1])  
 )  
 NodoControl.add\_child(*t*[3])  
 NodoControl.add\_child(ASTNode(*t*[6]))  
 *t*[0] = NodoControl  
  
  
*def* p\_funcion\_unset(*t*):  
 """funcion\_unset : RUNSET PARIZQ expresion PARDER"""  
 NodoUnset = ASTNode(**"unset"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]))  
 NodoUnset.add\_child(*t*[3])  
 *t*[0] = NodoUnset  
  
  
*def* p\_funcion\_print(*t*):  
 """funcion\_print : RPRINT PARIZQ expresion PARDER"""  
 NodoUnset = ASTNode(**"print"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]))  
 NodoUnset.add\_child(*t*[3])  
 *t*[0] = NodoUnset  
  
  
*def* p\_declaracion\_registros(*t*):  
 """declaracion\_registros : ID ASIG expresion"""  
 NodoDeclaracion = ASTNode(  
 **"declaracion\_asignacion"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1])  
 )  
 NodoDeclaracion.add\_child(ASTNode(*t*[1]))  
 NodoDeclaracion.add\_child(*t*[3])  
 *t*[0] = NodoDeclaracion  
  
  
*def* p\_declaracion\_arraymod(*t*):  
 """declaracion\_registros : otra ASIG expresion"""  
 NodoDeclaracion = ASTNode(  
 **"declaracion\_asignacion"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[2])  
 )  
 NodoDeclaracion.add\_child(*t*[1])  
 NodoDeclaracion.add\_child(*t*[3])  
 *t*[0] = NodoDeclaracion  
  
  
*def* p\_otra(*t*):  
 """otra : ID lista\_accesos"""  
 NodoModificacion = ASTNode(  
 **"modificar\_array"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1])  
 )  
 NodoModificacion.add\_child(ASTNode(*t*[1]))  
 NodoModificacion.add\_child(*t*[2])  
 *t*[0] = NodoModificacion  
  
  
*def* p\_expresion\_binaria(*t*):  
 """expresion : expresion MAS expresion  
 | expresion MENOS expresion  
 | expresion POR expresion  
 | expresion DIVIDIDO expresion  
 | expresion MODULO expresion  
 | expresion RAND expresion  
 | expresion ROR expresion  
 | expresion RXOR expresion  
 | expresion MAYOR expresion  
 | expresion MENOR expresion  
 | expresion MAYORIGUAL expresion  
 | expresion MENORIGUAL expresion  
 | expresion IGUAL expresion  
 | expresion DIFERENTE expresion  
 | expresion RANDBIT expresion  
 | expresion RORBIT expresion  
 | expresion RXORBIT expresion  
 | expresion SHIFTIZQ expresion  
 | expresion SHIFTDER expresion"""  
 *t*[0] = ASTNode(*t*[2], *t*.lexer.lineno, find\_column(entrada, *t*.slice[2]), *None*, *t*[1], *t*[3])  
  
  
*def* p\_casteos(*t*):  
 """casteos : PARIZQ tipo\_casteo PARDER expresion"""  
 NodoCasteo = ASTNode(**"casteos"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]))  
 NodoCasteo.add\_child(*t*[2])  
 NodoCasteo.add\_child(*t*[4])  
 *t*[0] = NodoCasteo  
  
  
*def* p\_tipo\_casteo(*t*):  
 """tipo\_casteo : RINT  
 | RFLOAT  
 | RCHAR """  
 NodoCasteo = ASTNode(**"tipo\_casteo"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]))  
 NodoauxCast = ASTNode(*t*[1], *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]))  
 NodoCasteo.add\_child(NodoauxCast)  
 *t*[0] = NodoCasteo  
  
  
*def* p\_valor\_absoluto(*t*):  
 """absValue : RABS PARIZQ expresion PARDER"""  
 NodoAbs = ASTNode(**"absoluto"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]))  
 NodoAbs.add\_child(*t*[3])  
 *t*[0] = NodoAbs  
  
  
*def* p\_lista\_de\_accesos(*t*):  
 """lista\_accesos : lista\_accesos CORIZQ expresion CORDER"""  
 NodoAuxLAcceso = *t*[1]  
 NodoRest = *t*[3]  
 NodoAuxLAcceso.add\_child(NodoRest)  
 *t*[0] = NodoAuxLAcceso  
  
  
*def* p\_l\_de\_accesos(*t*):  
 """lista\_accesos : CORIZQ expresion CORDER"""  
 NodoLAcceso = ASTNode(**"lista\_acceso"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]))  
 NodoLAcceso.add\_child(*t*[2])  
 *t*[0] = NodoLAcceso  
  
  
*def* p\_expresion\_unaria(*t*):  
 """expresion : MENOS expresion %prec UMENOS  
 | RNOT expresion  
 | RNOTBIT expresion"""  
 NodoExpresionUnaria = ASTNode(  
 **"expresion\_unaria"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1])  
 )  
 NodoExpresionUnaria.add\_child(ASTNode(*t*[1]))  
 NodoExpresionUnaria.add\_child(*t*[2])  
 *t*[0] = NodoExpresionUnaria  
  
  
*def* p\_expresion\_agrupacion(*t*):  
 "expresion : PARIZQ expresion PARDER"  
 NodoExpresionAgrupar = ASTNode(  
 **"exp\_agrupacion"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1])  
 )  
 NodoExpresionAgrupar.add\_child(*t*[2])  
 *t*[0] = NodoExpresionAgrupar  
  
  
*def* p\_expresion\_number(*t*):  
 """expresion : ENTERO  
 | DECIMAL  
 | ID  
 | CADENA  
 | RABS PARIZQ expresion PARDER  
 | RARRAY PARIZQ PARDER  
 | RREAD PARIZQ PARDER"""  
 *if t*[1] == **"abs"**:  
 *t*[0] = ASTNode(  
 **"abs"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]), *None*, *t*[3]  
 )  
 *else*:  
 *t*[0] = ASTNode(  
 **"expresion\_number"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1]), *None*, ASTNode(*t*[1])  
 )  
  
  
*def* p\_expresion\_casteo(*t*):  
 """expresion : casteos"""  
 NodoExpresionNumber = ASTNode(**"expresion\_number"**, *t*.lexer.lineno, 0)  
 NodoExpresionNumber.add\_child(*t*[1])  
 *t*[0] = NodoExpresionNumber  
  
  
*def* p\_exp\_accesoArray(*t*):  
 """expresion : accesos\_array"""  
 NodoExpresionNumber = ASTNode(**"expresion\_number"**, *t*.lexer.lineno, 0)  
 Nodoaux = *t*[1]  
 NodoExpresionNumber.add\_child(Nodoaux)  
 *t*[0] = NodoExpresionNumber  
  
  
*def* p\_acceso\_arrayExp(*t*):  
 """accesos\_array : ID lista\_accesos"""  
 NodoAccesArray = ASTNode(  
 **"access\_arrayexp"**, *t*.lexer.lineno, find\_column(entrada, *t*.slice[1])  
 )  
 NodoAccesArray.add\_child(ASTNode(*t*[1]))  
 NodoAccesArray.add\_child(*t*[2])  
 *t*[0] = NodoAccesArray  
  
  
*def* p\_error(*t*: LexToken):  
 descripcion = **"Error gramatical en: "** + *t*.value  
 ErrorSintactico = CError(  
 descripcion, *t*.lexer.lineno, find\_column(entrada, *t*), **"sintactico"** )  
 lErrores.append(ErrorSintactico)

IDE

“PyCharm 2019”

El software se llevó a cabo en PyCharm, ya que este IDE es un software capacitado para el uso del lenguaje python y especializado para aplicaciones, además de ser muy intuitivo con el lenguaje, proporciona recursos para la creación de interfaz gráfica donde el usuario puede crear una GUI amena y fácil para el usuario final.

Sistema Operativo

* MacOs 8.1 build 9600

Version de Python

